Table of Contents

Demonstration of controller synthesis via SReachPoint: Dubin's vehicle

This example will demonstrate the use of SReachTools in controller synthesis for a stochastic continuous-state discrete-time linear time-varying (LTV) systems.

Specifically, we will discuss how we can use SReachPoint to synthesize open-loop controllers and affine-disturbance feedback controllers for that maximize the probability of safety while respecting the system dynamics and control bounds. We demonstrate:

- `chance-open`: Chance-constrained approach that uses risk allocation and piecewise-affine approximations to formulate a linear program to synthesize an open-loop controller
- `genzps-open`: Fourier transforms that uses Genz's algorithm to formulate a nonlinear log-concave optimization problem to be solved using MATLAB's patternsearch to synthesize an open-loop controller
- `particle-open`: Particle control approach filter that formulates a mixed-integer linear program to synthesize an open-loop controller
- `chance-affine`: Chance-constrained approach that uses risk allocation and piecewise-affine approximations to formulate a difference-of-convex program to synthesize a closed-loop (affine disturbance feedback) controller. The controller synthesis is done by solving a series of second-order cone programs.

Our approaches are grid-free and recursion-free resulting in highly scalable solutions, especially for Gaussian-perturbed linear systems.

This script is part of the SReachTools toolbox, which is licensed under GPL v3 or (at your option) any later version. A copy of this license is given in https://github.com/unm-hscl/SReachTools/blob/master/LICENSE.

```
% Commands to ensure clean setup
close all;clc;clearvars;srtinit
```

Problem: Stochastic reachability of a target tube for a Dubin's vehicle

We consider a Dubin's vehicle with known turning rate sequence $\overline{\omega} = [\omega_0 \ \omega_1 \ \dots \ \omega_{T-1}]^\top \in R^T$, with additive Gaussian disturbance. The resulting dynamics are,

$$x_{k+1} = x_k + T_s \cos\left(\theta_0 + \sum_{i=1}^{k-1} \omega_i T_s\right) v_k + \eta_k^x$$

$$y_{k+1} = y_k + T_s \sin\left(\theta_0 + \sum_{i=1}^{k-1} \omega_i T_s\right) v_k + \eta_k^y$$

where x, y are the positions (state) of the Dubin's vehicle in x- and y- axes, v_k is the velocity of the vehicle (input), $\eta_k^{(\cdot)}$ is the additive Gaussian disturbance affecting the dynamics, T_s is the sampling time, and θ_0 is the initial heading direction. We define the disturbance as $[\eta_k^x \ \eta_k^y] \sim \mathcal{N}([00], 10^{-3} I_2)$.

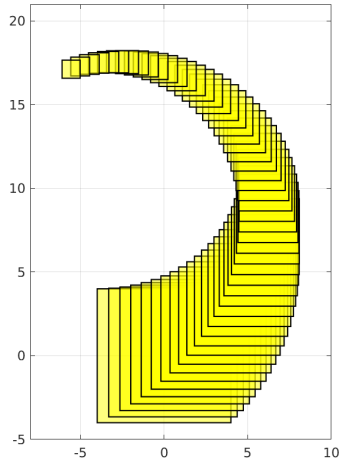
```
n_mcarlo_sims = 1e5;                            % Monte-Carlo simulation
 particles
sampling_time = 0.1;                            % Sampling time
init_heading = pi/10;                           % Initial heading
% Known turning rate sequence
time_horizon = 50;
omega = pi/time_horizon/sampling_time;
turning_rate = omega*ones(time_horizon,1);
% Input space definition
umax = 6;
input_space = Polyhedron('lb',0,'ub',umax);
% Disturbance matrix and random vector definition
dist_matrix = eye(2);
eta_dist = RandomVector('Gaussian',zeros(2,1), 0.001 * eye(2));

[sys, heading_vec] = getDubinsCarLtv('add-dist', turning_rate,
 init_heading, ...
    sampling_time, input_space, dist_matrix, eta_dist);
```

Target tube definition

We define the target tube to be a collection of boxes centered about the nominal trajectory with fixed velocity of $umax * 3/2$ (faster than the maximum velocity allowed) and the heading angle sequence with $\pi/2$ removed. The boxes have an exponentially decaying half-length with time constant as half of the time horizon.

```
box_halflength = 4;
v_nominal = umax * 3/2;
time_const = 1/2*time_horizon;
target_tube_cell = cell(time_horizon + 1,1);
```

```
figure(100);clf;hold on
angle_at_the_center = (heading_vec)-pi/2;
center_box = zeros(2, time_horizon + 1);
for itt = 0:time_horizon
    center_box(:, itt+1) = v_nominal *...
        [cos(angle_at_the_center(itt+1))-cos(angle_at_the_center(1));
         sin(angle_at_the_center(itt+1))-sin(angle_at_the_center(1))];
    target_tube_cell{itt+1} = Polyhedron(...
        'lb',center_box(:, itt+1) - box_halflength * exp(- itt/
time_const),...
        'ub', center_box(:, itt+1) + box_halflength*exp(- itt/
time_const));
    plot(target_tube_cell{itt+1},'alpha',0.5,'color','y');
end
axis equal
axis([-8    10   -5   21]);
box on;
grid on;
target_tube = Tube(target_tube_cell{:});
```



Specifying initial states and which options to run

```
chance_open_run = 0;
genzps_open_run = 0;
particle_open_run = 0;
chance_affine_run = 0;
% Initial states for each of the method
init_state_chance_open = [2;2] + [-1;1];
init_state_genzps_open = [2;2] + [1;-1];
init_state_particle_open = [2;2] + [0;1];
init_state_chance_affine = [2;2] + [2;1];
```

Quantities needed to compute the optimal mean trajectory

We first compute the dynamics of the concatenated state vector $X = Zx_0 + HU + GW$, and compute the concatentated random vector W and its mean.

```
[Z,H,G] = sys.getConcatMats(time_horizon);
% Compute the mean trajectory of the concatenated disturbance vector
muW = sys.dist.concat(time_horizon).parameters.mean;
```

SReachPoint: chance-open

This implements the chance-constrained approach to compute a globally opt open-loop controller. This approach uses risk allocation and piecewise-affine overapproximation of the inverse normal cumulative density function to formulate a linear program for this purpose. Naturally, this is one of the fastest ways to compute an open-loop controller and an underapproximative probabilistic guarantee of safety. However, due to the use of Boole's inequality for risk allocation, it provides a conservative estimate of safety using the open-loop controller.

```
if chance_open_run
    fprintf('\n\nSReachPoint with chance-open\n');
    % Set the maximum piecewise-affine overapproximation error to 1e-3
    opts = SReachPointOptions('term', 'chance-
open','pwa_accuracy',1e-3);
    tic;
    [prob_chance_open, opt_input_vec_chance_open] =
 SReachPoint('term', ...
        'chance-open', sys, init_state_chance_open, target_tube,
 opts);
    elapsed_time_chance_open = toc;
    if prob_chance_open
        % Optimal mean trajectory construction
        % mean_X = Z * x_0 + H * U + G * \mu_W
        opt_mean_X_chance_open = Z * init_state_chance_open + ...
            H * opt_input_vec_chance_open + G * muW;
        opt_mean_traj_chance_open =
 reshape(opt_mean_X_chance_open, ...
            sys.state_dim,[]);
        % Check via Monte-Carlo simulation
        concat_state_realization =
 generateMonteCarloSims(n_mcarlo_sims, ...
            sys, init_state_chance_open, time_horizon,...
            opt_input_vec_chance_open);
        mcarlo_result =
 target_tube.contains(concat_state_realization);
        simulated_prob_chance_open = sum(mcarlo_result)/n_mcarlo_sims;
    else
        simulated_prob_chance_open = NaN;
    end
    fprintf('SReachPoint underapprox. prob: %1.2f | Simulated prob:
 %1.2f\n',...
```

```
        prob_chance_open, simulated_prob_chance_open);
    fprintf('Computation time: %1.3f\n', elapsed_time_chance_open);
end
```

SReachPoint: `genzps-open`

This implements the Fourier transform-based approach to compute a globally opt open-loop controller. This approach uses Genz's algorithm to compute the probability of safety and optimizes the joint chance constraint involved in maximizing this probability. To handle the noisy behaviour of the Genz's algorithm, we rely on MATLAB's `patternsearch` for the nonlinear optimization. The global optity of the open-loop controller is guaranteed by the log-concavity of the problem. Internally, we use the `chance-open` to initialize the nonlinear solver. Hence, this approach will return an open-loop controller with safety at least as good as `chance-open`.

```
if genzps_open_run
    fprintf('\n\nSReachPoint with genzps-open\n');
    opts = SReachPointOptions('term', 'genzps-open', ...
        'PSoptions',psoptimset('display','iter'));
    tic
    [prob_genzps_open, opt_input_vec_genzps_open] =
 SReachPoint('term', ...
        'genzps-open', sys, init_state_genzps_open, target_tube,
 opts);
    elapsed_time_genzps = toc;
    if prob_genzps_open > 0
        % Optimal mean trajectory construction
        % mean_X = Z * x_0 + H * U + G * \mu_W
        opt_mean_X_genzps_open =  Z * init_state_genzps_open + ...
            H * opt_input_vec_genzps_open + G * muW;
        opt_mean_traj_genzps_open= reshape(opt_mean_X_genzps_open, ...
            sys.state_dim,[]);
        % Check via Monte-Carlo simulation
        concat_state_realization =
 generateMonteCarloSims(n_mcarlo_sims, ...
            sys, init_state_genzps_open, time_horizon,...
            opt_input_vec_genzps_open);
        mcarlo_result =
 target_tube.contains(concat_state_realization);
        simulated_prob_genzps_open = sum(mcarlo_result)/n_mcarlo_sims;
    else
        simulated_prob_genzps_open = NaN;
    end
    fprintf('SReachPoint underapprox. prob: %1.2f | Simulated prob:
 %1.2f\n',...
        prob_genzps_open, simulated_prob_genzps_open);
    fprintf('Computation time: %1.3f\n', elapsed_time_genzps);
end
```

SReachPoint: `particle-open`

This implements the particle control approach to compute an open-loop controller. This approach is a sampling-based technique and hence the resulting probability estimate is random with its variance going to zero as the number of samples considered goes to infinity. Note that since a mixed-integer linear program

is solved underneath with the number of binary variables corresponding to the number of particles, using too many particles can cause an exponential increase in computational time.

```
if particle_open_run
    fprintf('\n\nSReachPoint with particle-open\n');
    opts = SReachPointOptions('term','particle-open','verbose',1,...
        'num_particles',50);
    tic
    [prob_particle_open, opt_input_vec_particle_open] =
 SReachPoint('term', ...
        'particle-open', sys, init_state_particle_open, target_tube,
 opts);
    elapsed_time_particle = toc;
    if prob_particle_open > 0
        % Optimal mean trajectory construction
        % mean_X = Z * x_0 + H * U + G * \mu_W
        opt_mean_X_particle_open =  Z * init_state_particle_open + ...
            H * opt_input_vec_particle_open + G * muW;
        opt_mean_traj_particle_open =...
            reshape(opt_mean_X_particle_open, sys.state_dim,[]);
        % Check via Monte-Carlo simulation
        concat_state_realization =
 generateMonteCarloSims(n_mcarlo_sims, ...
            sys, init_state_particle_open,time_horizon,...
            opt_input_vec_particle_open);
        mcarlo_result =
 target_tube.contains(concat_state_realization);
        simulated_prob_particle_open = sum(mcarlo_result)/
n_mcarlo_sims;
    else
        simulated_prob_particle_open = NaN;
    end
    fprintf('SReachPoint approx. prob: %1.2f | Simulated prob: %1.2f
\n',...
        prob_particle_open, simulated_prob_particle_open);
    fprintf('Computation time: %1.3f\n', elapsed_time_particle);
end
```

SReachPoint: `chance-affine`

This implements the chance-constrained approach to compute a locally opt affine disturbance feedback controller. This approach uses risk allocation and piecewise-affine overapproximation of the inverse normal cumulative density function to formulate a difference-of-convex program. We utilize penalty convex-concave procedure to solve this program to a local optimum. Due to its construction of the affine feedback controller, this approach typically permits the construction of the highest underapproximative probability guarantee. Since affine disturbance feedback controllers can not satisfy hard control bounds, we relax the control bounds to be probabilistically violated with at most a probability of 0.01

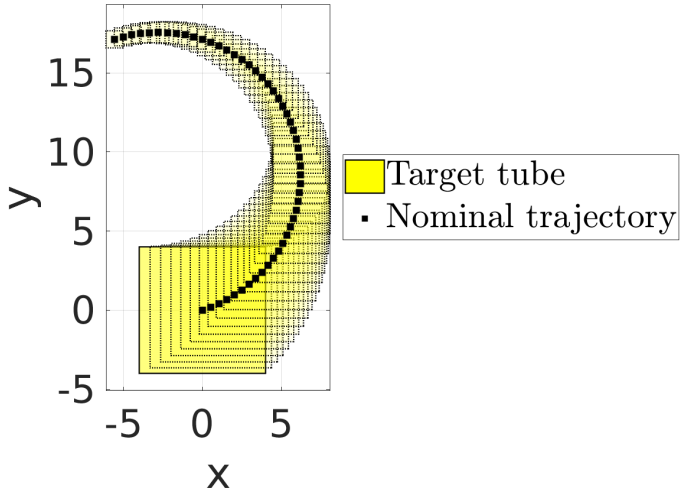
```
if chance_affine_run
    fprintf('\n\nSReachPoint with chance-affine\n');
    opts = SReachPointOptions('term', 'chance-affine',...
        'max_input_viol_prob', 1e-2, 'verbose',2);
    tic
    [prob_chance_affine, opt_input_vec_chance_affine,...
```

```
        opt_input_gain_chance_affine] = SReachPoint('term', 'chance-
affine',...
            sys, init_state_chance_affine, target_tube, opts);
    elapsed_time_chance_affine = toc;
    if prob_chance_affine > 0
        % mean_X = Z * x_0 + H * (M \mu_W + d) + G * \mu_W
        opt_mean_X_chance_affine = Z * init_state_chance_affine +...
            H * opt_input_vec_chance_affine + ...
            (H * opt_input_gain_chance_affine + G) * muW;
        % Optimal mean trajectory construction
        opt_mean_traj_chance_affine =
 reshape(opt_mean_X_chance_affine, ...
            sys.state_dim,[]);
        % Check via Monte-Carlo simulation
        concat_state_realization =
 generateMonteCarloSims(n_mcarlo_sims, ...
            sys, init_state_chance_affine, time_horizon,...
            opt_input_vec_chance_affine,
 opt_input_gain_chance_affine);
        mcarlo_result =
 target_tube.contains(concat_state_realization);
        simulated_prob_chance_affine = sum(mcarlo_result)/
n_mcarlo_sims;
    else
        simulated_prob_chance_affine = NaN;
    end
    fprintf('SReachPoint underapprox. prob: %1.2f | Simulated prob:
 %1.2f\n',...
        prob_chance_affine, simulated_prob_chance_affine);
    fprintf('Computation time: %1.3f\n', elapsed_time_chance_affine);
end
```

Plot of the optimal mean trajectories

```
figure(101);
clf;
hold on;
for itt = 0:time_horizon
    if itt==0
        % Remember the first the tube
        h_target_tube =
 plot(target_tube_cell{1},'alpha',0.5,'color','y');
    else
        plot(target_tube_cell{itt
+1},'alpha',0.08,'LineStyle',':','color','y');
    end
end
axis equal
h_nominal_traj = scatter(center_box(1,:), center_box(2,:),
 50,'ks','filled');
h_vec = [h_target_tube, h_nominal_traj];
legend_cell = {'Target tube', 'Nominal trajectory'};
% Plot the opt mean trajectory from the vertex under study
```

```
if chance_open_run
    h_opt_mean_ccc = scatter(...
          [init_state_chance_open(1),
 opt_mean_traj_chance_open(1,:)], ...
          [init_state_chance_open(2),
 opt_mean_traj_chance_open(2,:)], ...
          30, 'bo', 'filled','DisplayName', 'Mean trajectory (chance-
open)');
    legend_cell{end+1} = 'Mean trajectory (chance-open)';
    h_vec(end+1) = h_opt_mean_ccc;
end
if chance_affine_run
    h_opt_mean_chance_affine = scatter(...
          [init_state_chance_affine(1),
 opt_mean_traj_chance_affine(1,:)], ...
          [init_state_chance_affine(2),
 opt_mean_traj_chance_affine(2,:)], ...
          30, 'ms', 'filled','DisplayName', 'Mean trajectory (chance-
affine)');
    legend_cell{end+1} = 'Mean trajectory (chance-affine)';
    h_vec(end+1) = h_opt_mean_chance_affine;
end
if genzps_open_run
    h_opt_mean_genzps = scatter(...
          [init_state_genzps_open(1),
 opt_mean_traj_genzps_open(1,:)], ...
          [init_state_genzps_open(2),
 opt_mean_traj_genzps_open(2,:)], ...
          30, 'kd','DisplayName', 'Mean trajectory (genzps-open)');
    legend_cell{end+1} = 'Mean trajectory (genzps-open)';
    h_vec(end+1) = h_opt_mean_genzps;
end
if particle_open_run
    h_opt_mean_particle = scatter(...
          [init_state_particle_open(1),
 opt_mean_traj_particle_open(1,:)], ...
          [init_state_particle_open(2),
 opt_mean_traj_particle_open(2,:)], ...
          30, 'r^', 'filled','DisplayName', 'Mean trajectory
 (particle-open)');
    legend_cell{end+1} = 'Mean trajectory (particle-open)';
    h_vec(end+1) = h_opt_mean_particle;
end
legend(h_vec,
 legend_cell, 'Location','EastOutside', 'interpreter','latex');
xlabel('x');
ylabel('y');
axis equal
box on;
set(gca,'FontSize',30);
```



Published with MATLAB® R2017b